EUROPA Quick Start

This tutorial provides just enough information for you to create and run simple planning problems using EUROPA. Once you're familiar with the basics, see the Documentation page for more details.

EUROPA can be used to solve Constraint Programming, Scheduling and Planning problems. For the purposes of this Quick Start guide we will use a simple planning problem.

These are the main steps that a EUROPA user will normally follow to solve a problem :

- Create a model of the problem domain using NDDL, EUROPA's modeling language
- Create a particular instance of the problem using NDDL
- Instantiate a solver and ask it to look for a solution
- Inspect the plan, possibly modify the problem, tune the solver and re-plan
- When the model and the solver are working as expected, probably embed them into a larger application

To try the steps below yourself, you must have already installed EUROPA.

Now, for our example, let's assume that we will create a plan to turn a light bulb on and off.

For this simple domain here is what we would do (the files for this example can be found here).

Create a NDDL model for the domain

The New Domain Description Language (NDDL) is a simple but powerful language used to describe both problem domains and problem instances in EUROPA

```
// Light-model.nddl

#include "Plasma.nddl"

class LightSwitch extends Timeline
{
    predicate turnOn  { duration=1; }
    predicate turnOff { duration=1; }
}

class LightBulb extends Timeline
{
    LightSwitch mySwitch_;

    LightBulb(LightSwitch b)
    {
        mySwitch_ = b;
    }
```

```
    predicate On  {}
    predicate Off {}
}

LightBulb::On
{
    met_by(object.Off);               // Bulb must be Off to be turned On
    met_by(object.mySwitch_.turnOn); // Must be turned on through the switch
}

LightBulb::Off
{
    met_by(object.On);                // Bulb must be On to be turned Off
    met_by(object.mySwitch_.turnOff); // Must be turned off through the switch
}
```

Create a problem instance

```
// Light-initial-state.nddl

#include "Light-model.nddl"

// Problem instance : turning the light off

// We have one bulb with its corresponding switch :
LightSwitch switch1 = new LightSwitch();
LightBulb bulb1 = new LightBulb(switch1);

// At time 0, the bulb is on
fact(bulb1.On initialCondition);
eq(initialCondition.start,0);

// We want the bulb to be off by time 10
goal(bulb1.Off goal1);
lt(0,goal1.start);
lt(goal1.start,10);
```

Instantiate and run a Solver

EUROPA comes with a simple script called makeproject which generates the minimal C++ and Java applications to run a planner on your model. You can then tailor those applications to your own purposes, or use them as an example to embed EUROPA into your own application.

```
% cd $PLASMA_HOME
% ./makeproject Light ~
```

Replace the default model and initial state generated by makeproject with the model and initial state for the Light problem as seen above :

```
% cp $PLASMA_HOME/Examples/Light/*.nddl ~/Light
```

Let's run the Java application generated by makeproject :

```
% cd ~/Light
```

```
% ant
Buildfile: build.xml

init:

compile:

run:
     [echo] Running Light project
     [java]  autoWrite 0
```

The Light model is part of the EUROPA distribution, if you don't want to create your own project to try this, you can run it directly from the EUROPA directories :

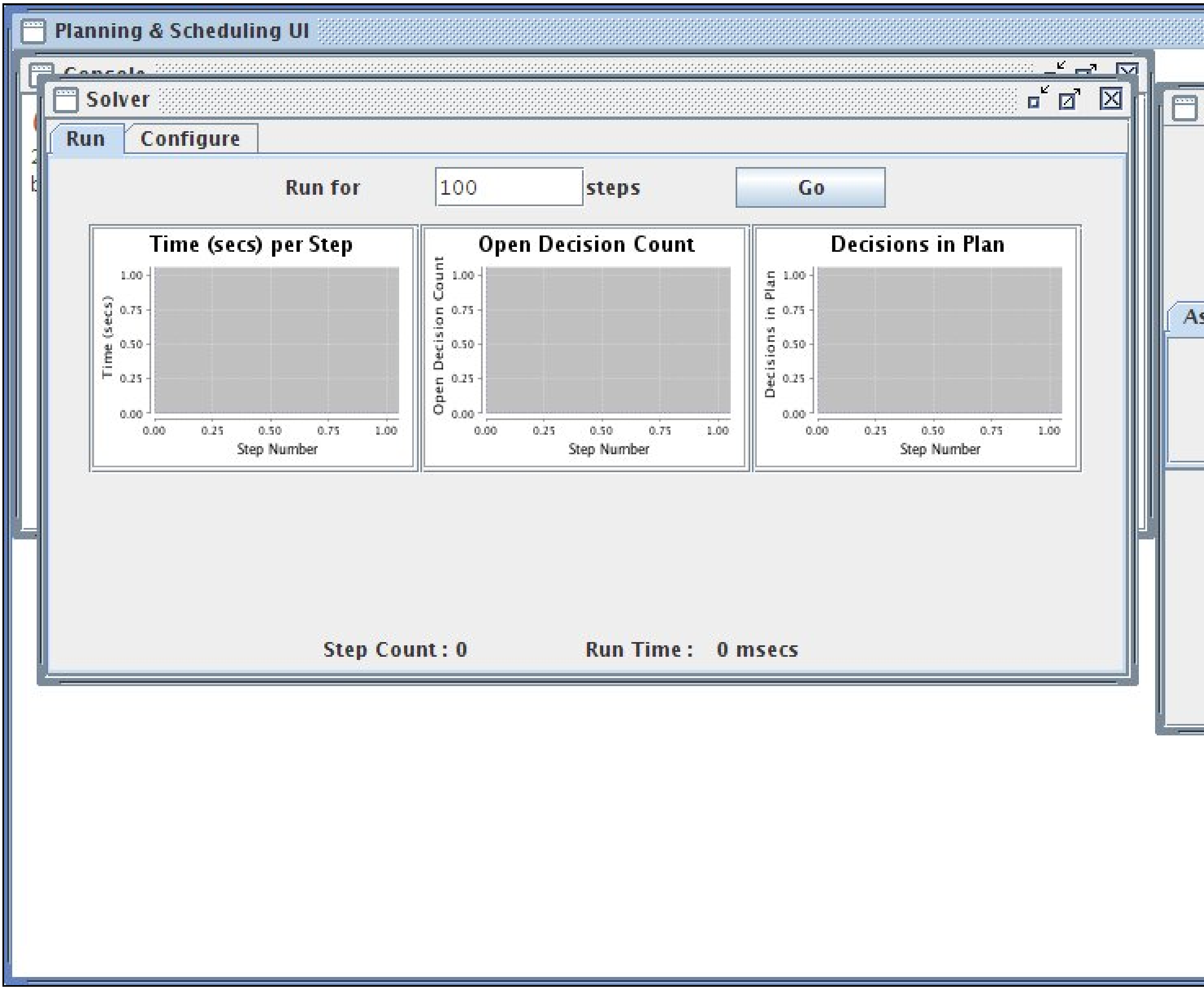
```
% cd $PLASMA_HOME/Examples/Light
% ant
Buildfile: build.xml

init:

compile:

run:
     [echo] Running Light project
     [java]  autoWrite 0
     [java] [XMLInterpreter:InterpretedObject]Initialized variable:bulb1.mySwitch_ to OBJECT-LightSwitc
```

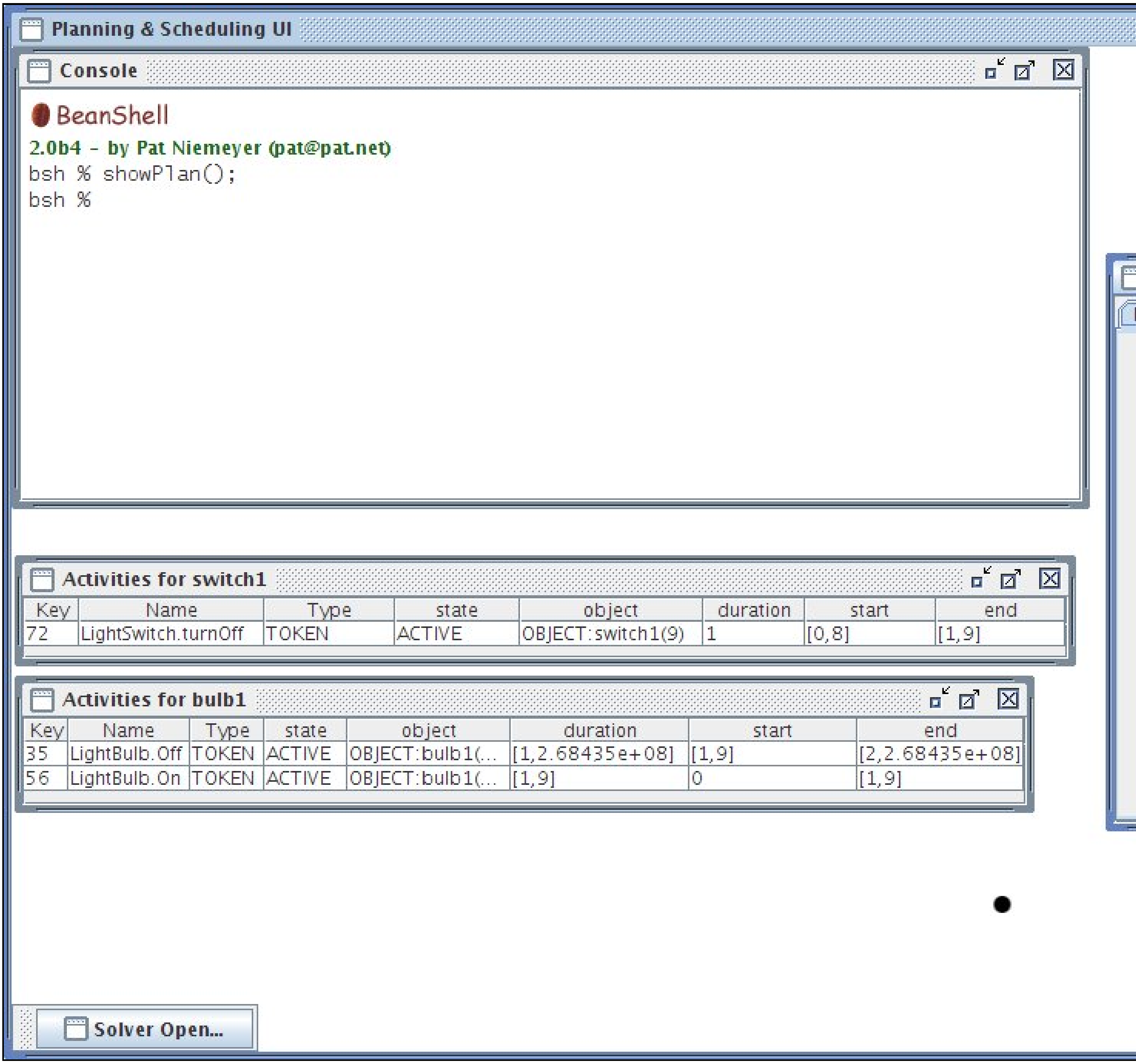
You should see the following window come up :



In the "Solver" window, hit the "Go" button to have the solver run. The solver should finish after 7 steps, satisfying the goal that was specified in Light-initial-state.nddl (that is, to turn the light off by time=10).

Examine Results

You can now run the method showPlan() defined in Light.bsh



You can now see that the planner decided that :

- bulb1 is On from [0] to [1,9] and Off from [1,9] to [infinity]
- lightSwitch1 is turned off at time [0,8]

The intervals mean that the action or state change could happen at any point in the interval, so for instance, "lightSwitch1 is turned off at time [0,8]" means that lightSwitch1 could be turned off at time 0, or at time 1, ..., or at time 8. You can modify your model or EUROPA's configuration to generate grounded plans (where all the values are points, instead of intervals), if that's what you want for your particular application.

Embed your model into an application

Sometimes, what we have done so far is all you may want to do : create a model, a problem instance, generate a plan, visualize the results, and perhaps dump them to a file.
In other cases you may want to embed this model as part of a bigger application, if that is the case, you will need to link in EUROPA as a library into your application and use the programmatic API (Java or C++) to perform the steps described above, you can see how this is done for this particular example in C++ and Java.
In the Java code, the main() method exposes an instance of EUROPA to BeanShell through the variable psengine, which is then driven through scripting to load the model, run the solver, etc. It could all be done in the Java main() as well, but scripting is much more useful as it allows you to play with EUROPA interactively.

The details of how to embed EUROPA into your application are provided here

Next Steps

- See the Examples page to see more advanced examples that can serve as starting points for your own model, or just to learn more about what EUROPA can do.
- Read a high-level overview of the planning components in EUROPA.
- Learn the details about modeling in NDDL, configuring the built-in solver and using EUROPA's debugging tools.
- How to create your own project
- How to embed your model into a C++ or Java application
- How to extend EUROPA by adding your own constraints, solvers, etc